autogluon_each_location

October 28, 2023

1 Config

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*10
presets = "best_quality"#'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

drop_night_outliers = True
drop_null_outliers = False

# to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
↪gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa", ␣
↪"pressure_100m:hPa"]
to_drop = ["wind_speed_w_1000hPa:ms", "wind_speed_u_10m:ms", "wind_speed_v_10m:
↪ms"]


use_groups = False
n_groups = 8

# auto_stack = True
num_stack_levels = 0
num_bag_folds = None# 8
num_bag_sets = None#20

use_tune_data = True
```

```
use_test_data = True
#tune_and_test_length = 0.5 # 3 months from end
# holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False
```

2 Loading and preprocessing

[2]:
```
import pandas as pd
import numpy as np




import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()

    # Create a set for constant-time lookup
    index_set = set(X.index)
```

```
    # Vectorized time shifting
    one_hour = pd.Timedelta('1 hour')
    shifted_indices = X_shifted.index + one_hour
    X_shifted.loc[shifted_indices.isin(index_set)] = X.
 ↪loc[shifted_indices[shifted_indices.isin(index_set)]][columns]

    # set last row to same as second last row
    X_shifted.iloc[-1] = X_shifted.iloc[-2]

    # Count
    count1 = len(shifted_indices[shifted_indices.isin(index_set)])
    count2 = len(X_shifted) - count1

    print("COUNT1", count1)
    print("COUNT2", count2)

    # Rename columns
    X_old_unshifted = X_shifted.copy()
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
 ↪columns]

    date_calc = None
    # If 'date_calc' is present, handle it
    if 'date_calc' in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']




    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
```

```
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X




def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train





def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
 ↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
 ↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
 ↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
```

```
        X_test["estimated_diff_hours"] = (X_test.index - pd.
↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1


    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    print(f"Size of estimated after dropping nans:␣
↪{len(X_train[X_train['is_estimated']==1].dropna(subset=['y']))}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
```

```python
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Size of estimated after dropping nans: 4418
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
```

```
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Size of estimated after dropping nans: 3625
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
Size of estimated after dropping nans: 2954
```

2.1 Feature enginering

2.1.1 Remove anomalies

[3]:
```python
import numpy as np
import pandas as pd


# loop thorugh x train[y], keep track of streaks of same values and replace␣
↪them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
        allowed = [0]
        found_streaks = {}
```

```
        for idx in x.index:
            value = x[idx]
            # if location == "B":
            #     continue

            if value == last_val and value not in allowed:
                streak_length += 1
                streak_indices.append(idx)
            else:
                streak_length = 1
                last_val = value
                streak_indices.clear()

            if streak_length > max_streak_length:
                found_streaks[value] = streak_length

                for streak_idx in streak_indices:
                    x[idx] = np.nan
                streak_indices.clear()  # clear after setting to NaN to avoid␣
↪setting multiple times
        df.loc[df["location"] == location, column] = x

        print(f"Found streaks for location {location}: {found_streaks}")

    return df


# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")
```

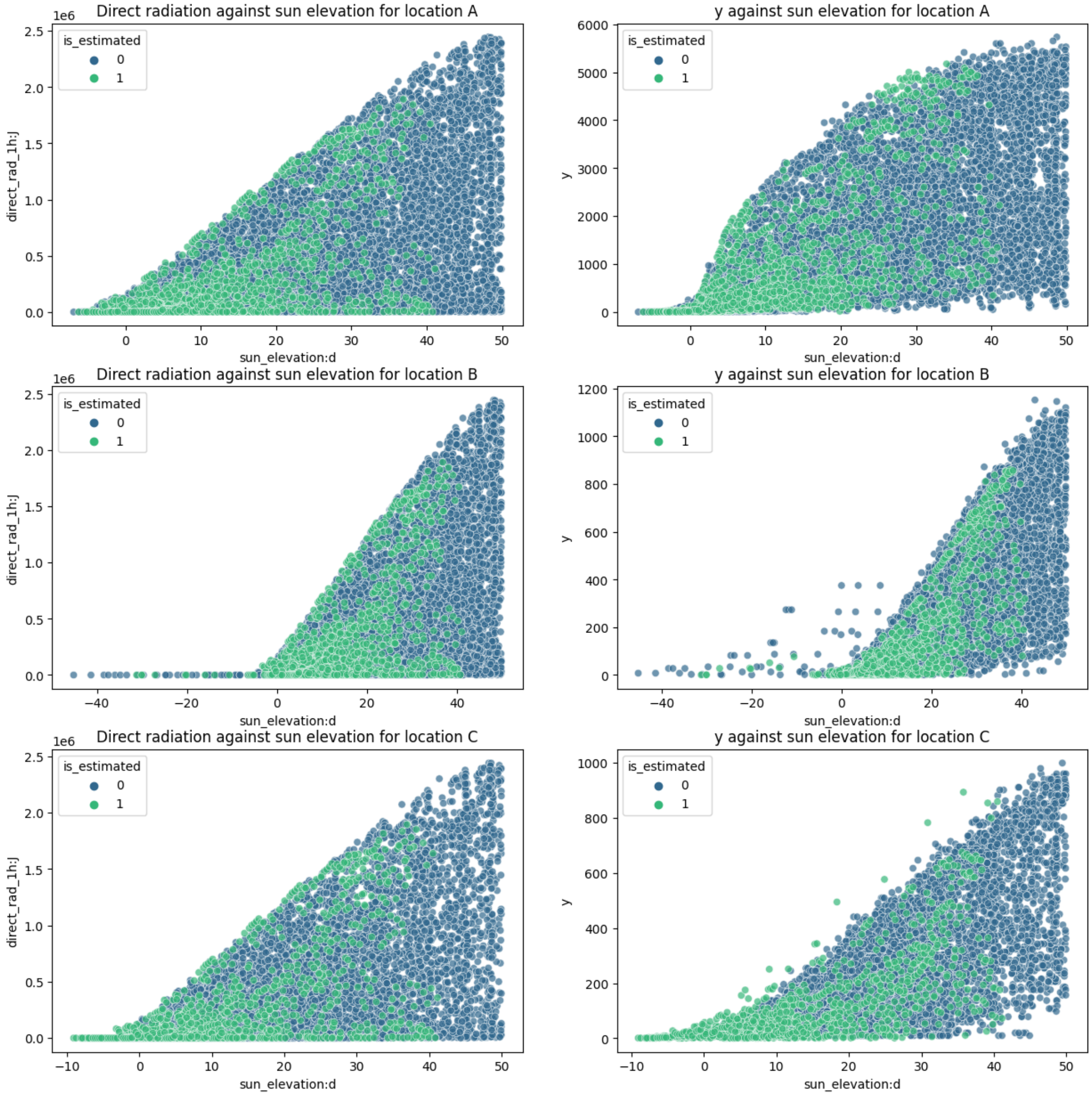
```
Found streaks for location A: {}
Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78,
18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7,
79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997:
55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375:
202, 2.5875: 12, 81.075: 210}
Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}
```

[4]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

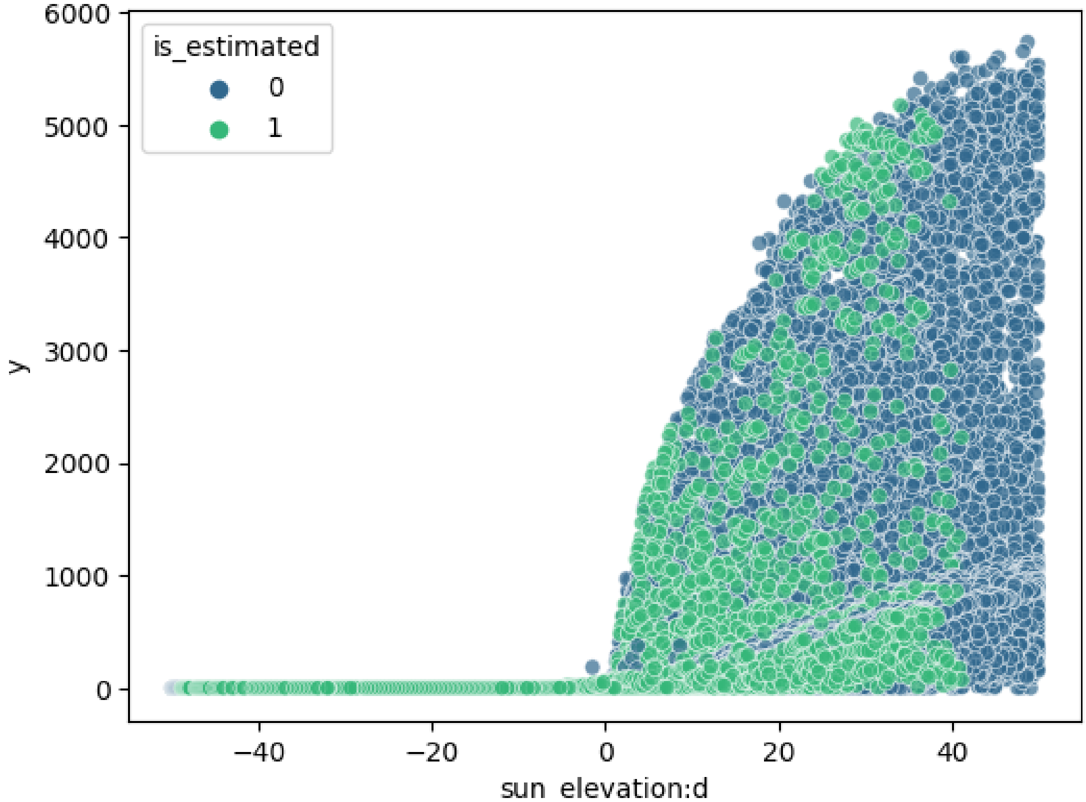
```
Dropped rows:  9293
```

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]

# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location], x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location], x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()
```

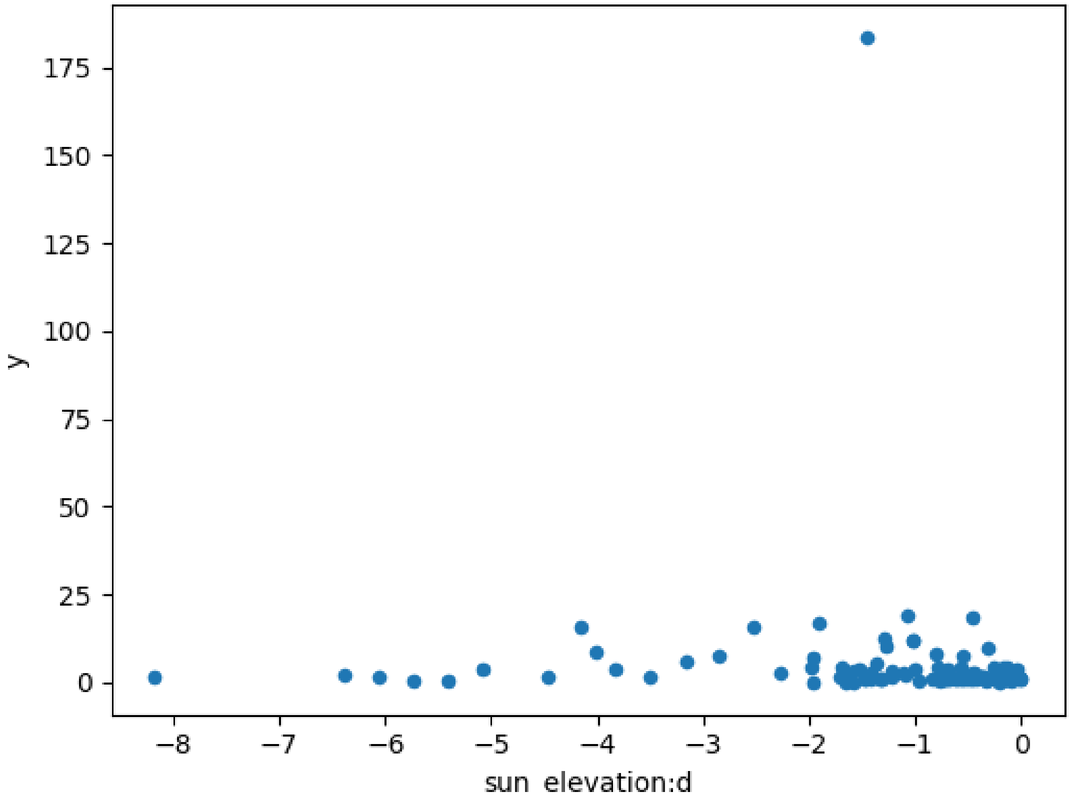


[6]:
```python
thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=
 ↪thresh) & (X_train["y"] >= 0.1)
if drop_night_outliers:
    X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",
 ↪palette="viridis", alpha=0.7)
plt.show()
```



[7]:
```python
# location B count number of rows with y > 0 and sun_elevation:d < 0

condition = (X_train["location"] == "B") & (X_train["y"] > 0) &␣
 ↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

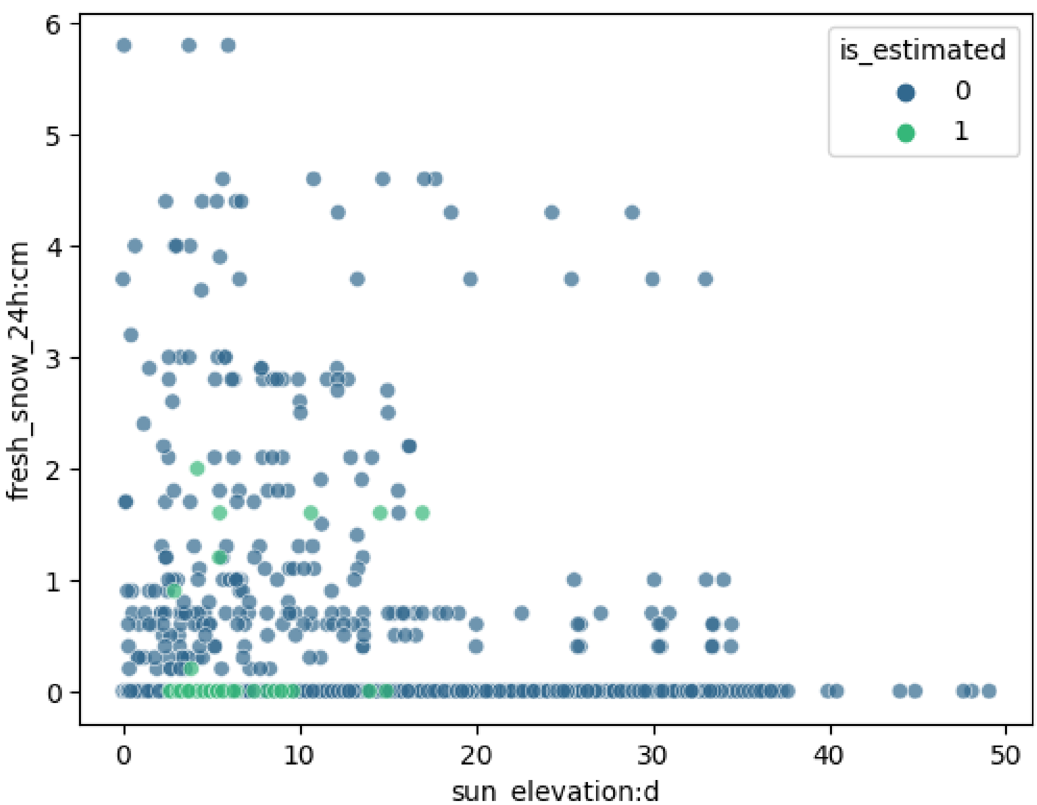
[7]: <AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>



[8]:
```
# set y to nan where y is 0, but direct_rad_1h:J or diffuse_rad_1h:J are > 0␣
↪(or some threshold)
threshold_direct = X_train["direct_rad_1h:J"].max() * 0.01
threshold_diffuse = X_train["diffuse_rad_1h:J"].max() * 0.01
print(f"Threshold direct: {threshold_direct}")
print(f"Threshold diffuse: {threshold_diffuse}")

mask = (X_train["y"] == 0) & ((X_train["direct_rad_1h:J"] > threshold_direct) |␣
↪(X_train["diffuse_rad_1h:J"] > threshold_diffuse)) & (X_train["sun_elevation:
↪d"] > 0) & (X_train["fresh_snow_24h:cm"] < 6) & (X_train[['fresh_snow_12h:
↪cm', 'fresh_snow_1h:cm',  'fresh_snow_3h:cm', 'fresh_snow_6h:cm']].
↪sum(axis=1) == 0)
print(len(X_train[mask]))

#print(X_train[mask][[x for x in X_train.columns if "snow" in x]])

# show plot where mask is true
#sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="y",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
```

```python
sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="fresh_snow_24h:cm",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
plt.show()

#sns.scatterplot(data=X_train[mask], x="fresh_snow_24h:cm",␣
↪y="total_cloud_cover:p", hue="is_estimated", palette="viridis", alpha=0.7)



# set y to nan where mask
if drop_null_outliers:
    X_train.loc[mask, "y"] = np.nan

# show how many rows for each location, and for estimated and not estimated
X_train[mask].groupby(["location", "is_estimated"]).count()["direct_rad_1h:J"]
```

```
Threshold direct: 24458.97
Threshold diffuse: 11822.505000000001
2599
```



[8]:
```
location  is_estimated
A         0                 87
          1                 10
B         0               1250
          1                 32
C         0               1174
          1                 46
Name: direct_rad_1h:J, dtype: int64
```

[9]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
 ↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  1876
```

2.1.2 Other stuff

[10]:
```
import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())


# If the "sample_weight" column is present and weight_evaluation is True,␣
 ↪multiply sample_weight with sample_weight_may_july if the ds is between␣
 ↪05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to␣
 ↪X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=␣
 ↪sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
 ↪((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))
```

```
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
 ↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
 ↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())






X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3              7.625
air_density_2m:kgm3                  1.2215
ceiling_height_agl:m            3644.050049
clear_sky_energy_1h:J            2896336.75
clear_sky_rad:W                  753.849976
cloud_base_agl:m                3644.050049
dew_or_rime:idx                         0.0
```

```
dew_point_2m:K                      280.475006
diffuse_rad:W                       127.475006
diffuse_rad_1h:J                    526032.625
direct_rad:W                             488.0
direct_rad_1h:J                    1718048.625
effective_cloud_cover:p              18.200001
elevation:m                                6.0
fresh_snow_12h:cm                          0.0
fresh_snow_1h:cm                           0.0
fresh_snow_24h:cm                          0.0
fresh_snow_3h:cm                           0.0
fresh_snow_6h:cm                           0.0
is_day:idx                                 1.0
is_in_shadow:idx                           0.0
msl_pressure:hPa                   1026.775024
precip_5min:mm                             0.0
precip_type_5min:idx                       0.0
pressure_100m:hPa                  1013.599976
pressure_50m:hPa                   1019.599976
prob_rime:p                                0.0
rain_water:kgm2                            0.0
relative_humidity_1000hPa:p          53.825001
sfc_pressure:hPa                   1025.699951
snow_density:kgm3                          NaN
snow_depth:cm                              0.0
snow_drift:idx                             0.0
snow_melt_10min:mm                         0.0
snow_water:kgm2                            0.0
sun_azimuth:d                       222.089005
sun_elevation:d                      44.503498
super_cooled_liquid_water:kgm2             0.0
t_1000hPa:K                         286.700012
total_cloud_cover:p                  18.200001
visibility:m                          52329.25
wind_speed_10m:ms                          2.6
wind_speed_u_10m:ms                       -1.9
wind_speed_v_10m:ms                      -1.75
wind_speed_w_1000hPa:ms                    0.0
is_estimated                                 0
y                                      4367.44
location                                     A
Name: 2019-06-11 13:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 23:00:00                       7.7               1.2235

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
```

```
2019-06-02 23:00:00          1689.824951                    0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 23:00:00              0.0       1689.824951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
2019-06-02 23:00:00      280.299988            0.0               0.0  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 23:00:00   286.899994                100.0  33770.648438

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 23:00:00               3.35                -3.35

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 23:00:00                0.275                      0.0

                     is_estimated    y  location
ds
2019-06-02 23:00:00             0  0.0         A

[1 rows x 48 columns]
```

[11]:
```
# Create a plot of X_train showing its "y" and color it based on the value of␣
↪the sample_weight column.
if "sample_weight" in X_train.columns:
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",␣
↪palette="deep", size=3)
    plt.show()
```

[12]:
```
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
```

```python
def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output␣
↪dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
```

```
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[13]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
estimated_set = TabularDataset(data[data["ds"] >= split_time]) # only estimated

test_set = pd.DataFrame()
tune_set = pd.DataFrame()
new_train_set = pd.DataFrame()

if not use_tune_data:
    raise Exception("Not implemented")

for location in locations:
    loc_data = data[data["location"] == location]
    num_train_rows = len(loc_data)

    tune_rows = 1500.0 # 2500.0
    if use_test_data:
        tune_rows = 1880.0#max(3000.0,␣
↪len(estimated_set[estimated_set["location"] == location]))
```

```
    holdout_frac = max(0.01, min(0.1, tune_rows / num_train_rows)) *␣
↪num_train_rows / len(estimated_set[estimated_set["location"] == location])

    print(f"Size of estimated for location {location}:␣
↪{len(estimated_set[estimated_set['location'] == location])}. Holdout frac␣
↪should be % of estimated: {holdout_frac}")

    # shuffle and split data
    loc_tune_set, loc_new_train_set =␣
↪split_and_shuffle_data(estimated_set[estimated_set['location'] == location],␣
↪40, holdout_frac)
    print(f"Length of location tune set : {len(loc_tune_set)}")
    new_train_set = pd.concat([new_train_set, loc_new_train_set])

    if use_test_data:
        loc_test_set, loc_tune_set = split_and_shuffle_data(loc_tune_set, 40, 0.
↪2)
        test_set = pd.concat([test_set, loc_test_set])

    tune_set = pd.concat([tune_set, loc_tune_set])




print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = tune_set

# number of rows in tuning data for each location
print("Shapes of tuning data", tuning_data.groupby('location').size())



if use_test_data:
    test_data = test_set
    print("Shape of test", test_data.shape[0])
```

```
train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    # ensure sample weights for data sum to the number of rows in the tuning /
↪train data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
tuning_data = TabularDataset(tuning_data)

if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Size of estimated for location A: 4214. Holdout frac should be % of estimated:
0.4461319411485524
Length of location tune set : 1846
Size of estimated for location B: 3533. Holdout frac should be % of estimated:
0.5321256722332296
Length of location tune set : 1846
Size of estimated for location C: 2923. Holdout frac should be % of estimated:
0.6431748203900103
Length of location tune set : 1841
Length of train set before adding test set 77247
Length of train set after adding test set 82384
Shapes of tuning data location
A    1485
B    1485
C    1481
dtype: int64
Shape of test 1082
```

3 Quick EDA

[14]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
↪label="y", sample=None)
```

[15]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

[16]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 120
Now creating submission number: 121
New filename: submission_121
```

[17]:
```
predictors = [None, None, None]
```

[18]:
```
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
```

```
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
↪drop(columns=["ds"]),
        time_limit=time_limit,
        presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_121_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   165.19 GB / 315.93 GB (52.3%)
Train Data Rows:    30934
Train Data Columns: 44
```

```
Tuning Data Rows:    1485
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 673.41535, 1195.24)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132193.01 MB
        Train Data (Original)  Memory Usage: 13.03 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…

Training model for location A…

        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.2s = Fit runtime
```

```
        41 features in original data used to generate 41 features in processed
data.
        Train Data (Processed) Memory Usage: 10.18 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.18s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.82s of the
599.82s of remaining time.
        -191.231         = Validation score   (-mean_absolute_error)
        0.04s    = Training   runtime
        0.4s     = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.28s of the
599.28s of remaining time.
        -192.9182        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.39s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 598.78s of the
598.78s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -85.9426         = Validation score   (-mean_absolute_error)
```

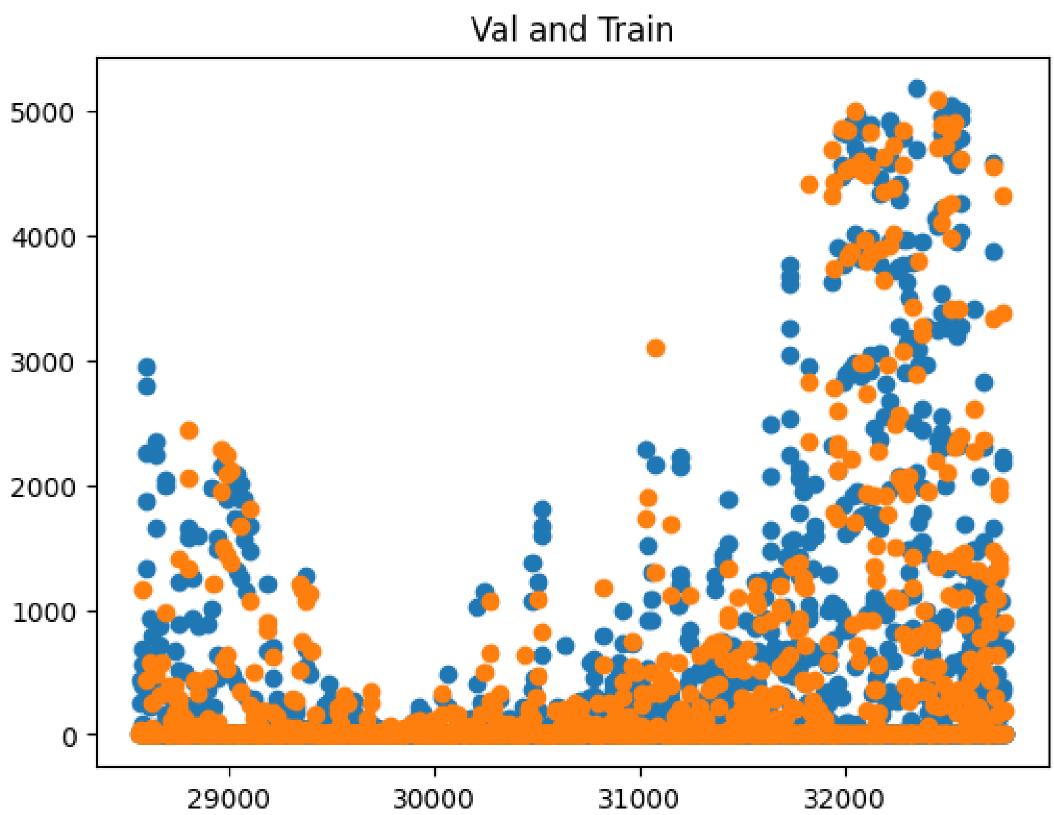
```
        32.39s   = Training   runtime
        12.57s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 557.46s of the
557.46s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -90.5139         = Validation score   (-mean_absolute_error)
        28.91s   = Training   runtime
        10.07s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 524.19s of
the 524.19s of remaining time.
        -98.6757         = Validation score   (-mean_absolute_error)
        8.49s    = Training   runtime
        1.14s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 513.42s of the
513.42s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -97.5224         = Validation score   (-mean_absolute_error)
        201.93s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 310.37s of the
310.37s of remaining time.
        -102.5531        = Validation score   (-mean_absolute_error)
        1.81s    = Training   runtime
        1.11s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 306.32s of
the 306.31s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.8453        = Validation score   (-mean_absolute_error)
        37.38s   = Training   runtime
        0.51s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 266.27s of the
266.27s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -94.785  = Validation score   (-mean_absolute_error)
        10.58s   = Training   runtime
        0.44s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 253.66s of the
253.66s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -88.0553         = Validation score   (-mean_absolute_error)
        97.11s   = Training   runtime
        0.43s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 155.02s of the
```

```
155.02s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -87.4068         = Validation score   (-mean_absolute_error)
        105.83s  = Training   runtime
        22.56s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
38.59s of remaining time.
        -82.4333         = Validation score   (-mean_absolute_error)
        0.45s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 561.89s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_121_A/")
Evaluation: mean_absolute_error on test data: -106.03254196629796
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -106.03254196629796,
    "root_mean_squared_error": -339.00796112150954,
    "mean_squared_error": -114926.39770376294,
    "r2": 0.8196327118009928,
    "pearsonr": 0.9100413662664941,
    "median_absolute_error": -2.631303310394287
}

Evaluation on test data:
-106.03254196629796
```

[19]:

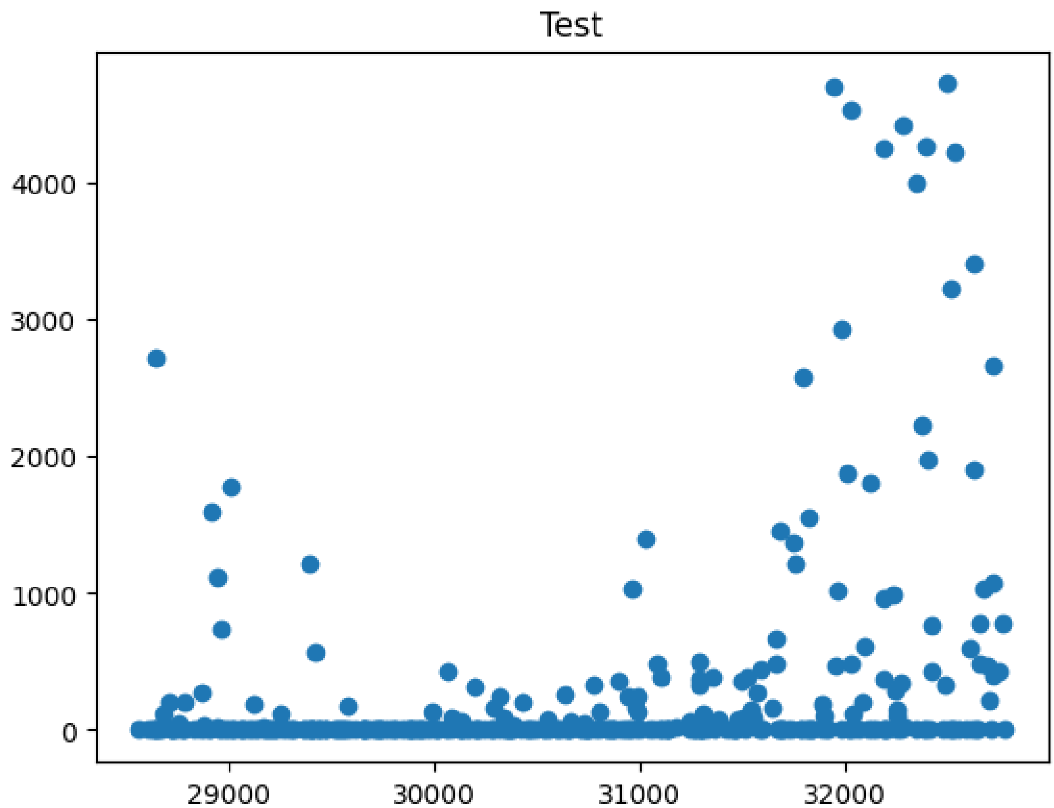
```
import matplotlib.pyplot as plt
leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_tune_data:
        plt.scatter(train_data[(train_data["location"] == loc) &␣
 ↪(train_data["is_estimated"]==True)]["y"].index,␣
 ↪train_data[(train_data["location"] == loc) &␣
 ↪(train_data["is_estimated"]==True)]["y"])
        plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.title("Val and Train")
        plt.show()

        if use_test_data:
            lb = predictors[i].leaderboard(test_data[test_data["location"] ==␣
 ↪loc])
```

```
            lb["location"] = loc
            plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
↪test_data[test_data["location"] == loc]["y"])
            plt.title("Test")

            return lb

    return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```




```
                     model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0       WeightedEnsemble_L2 -106.032542  -82.433295        4.306949
35.565912  235.787821                 0.003423                0.000628
0.449239            2       True          12
1         LightGBMXT_BAG_L1 -106.333520  -85.942583        0.885967
12.571410   32.394000                 0.885967               12.571410
32.394000             1       True           3
2     NeuralNetTorch_BAG_L1 -111.975066  -88.055325        0.197554
```

```
0.434585   97.114829                 0.197554                 0.434585
97.114829            1       True          10
3      LightGBMLarge_BAG_L1 -117.227800  -87.406820        3.220004
22.559289  105.829753                 3.220004                22.559289
105.829753            1       True          11
4           LightGBM_BAG_L1 -118.140013  -90.513901        0.714124
10.074449   28.912845                 0.714124                10.074449
28.912845            1       True           4
5    NeuralNetFastAI_BAG_L1 -118.158914 -103.845283        0.186673
0.512970   37.384766                 0.186673                 0.512970
37.384766            1       True           8
6           CatBoost_BAG_L1 -118.535034  -97.522393        0.060600
0.087009  201.925200                 0.060600                 0.087009
201.925200            1       True           6
7            XGBoost_BAG_L1 -122.725794  -94.785007        0.156679
0.439781   10.583787                 0.156679                 0.439781
10.583787            1       True           9
8      ExtraTreesMSE_BAG_L1 -130.386831 -102.553116        0.575443
1.106949    1.813190                 0.575443                 1.106949
1.813190            1       True           7
9    RandomForestMSE_BAG_L1 -130.657310  -98.675706        0.571127
1.135085    8.487255                 0.571127                 1.135085
8.487255            1       True           5
10    KNeighborsDist_BAG_L1 -189.567130 -192.918160        0.012731
0.394565    0.034994                 0.012731                 0.394565
0.034994            1       True           2
11    KNeighborsUnif_BAG_L1 -191.283846 -191.231007        0.152649
0.402471    0.035821                 0.152649                 0.402471
0.035821            1       True           1
```




```
[20]: loc = "B"
      predictors[1] = fit_predictor_for_location(loc)
      leaderboards[1] = leaderboard_for_location(1, loc)
```

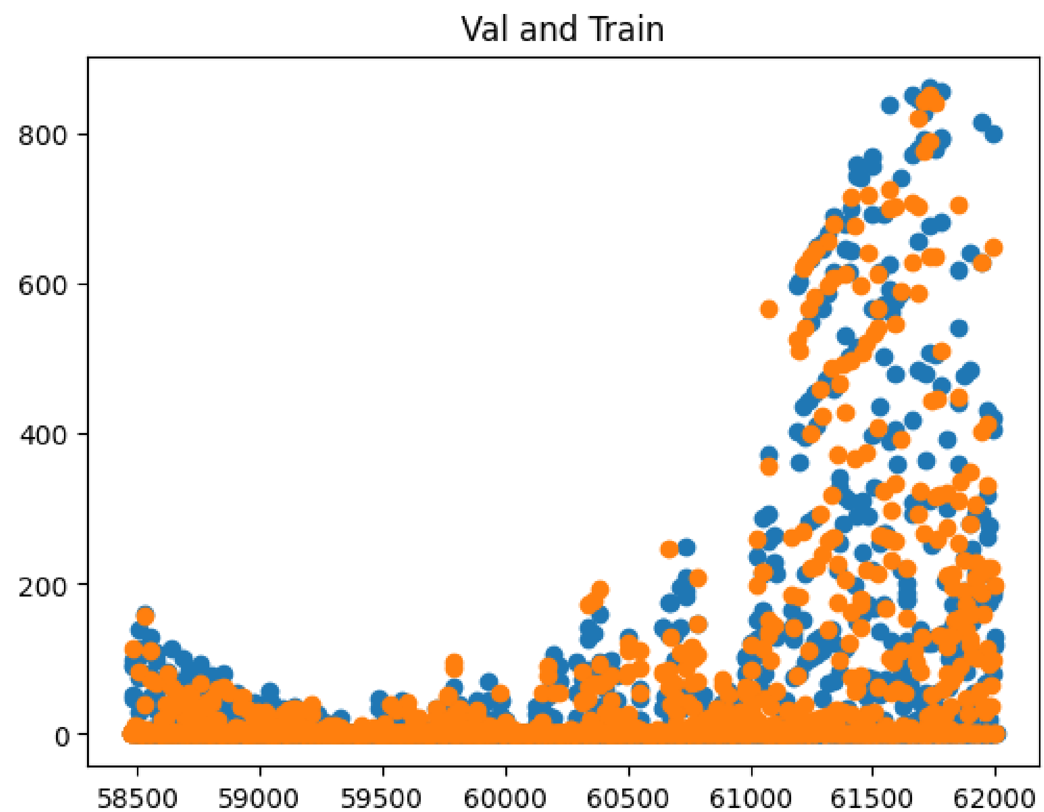
```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_121_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   162.93 GB / 315.93 GB (51.6%)
Train Data Rows:    27377
Train Data Columns: 44
Tuning Data Rows:    1485
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
```

```
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 98.11625, 206.48535)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …

Training model for location B…

Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130216.04 MB
        Train Data (Original)  Memory Usage: 11.6 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 41 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.2s = Fit runtime
        42 features in original data used to generate 42 features in processed
data.
        Train Data (Processed) Memory Usage: 9.29 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.21s …
AutoGluon will gauge predictive performance using evaluation metric:
```

```
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.79s of the
599.79s of remaining time.
        -28.5444         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.34s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.35s of the
599.35s of remaining time.
        -28.798  = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.36s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 598.89s of the
598.89s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.5683         = Validation score   (-mean_absolute_error)
        31.0s    = Training   runtime
        15.17s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 563.46s of the
563.46s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

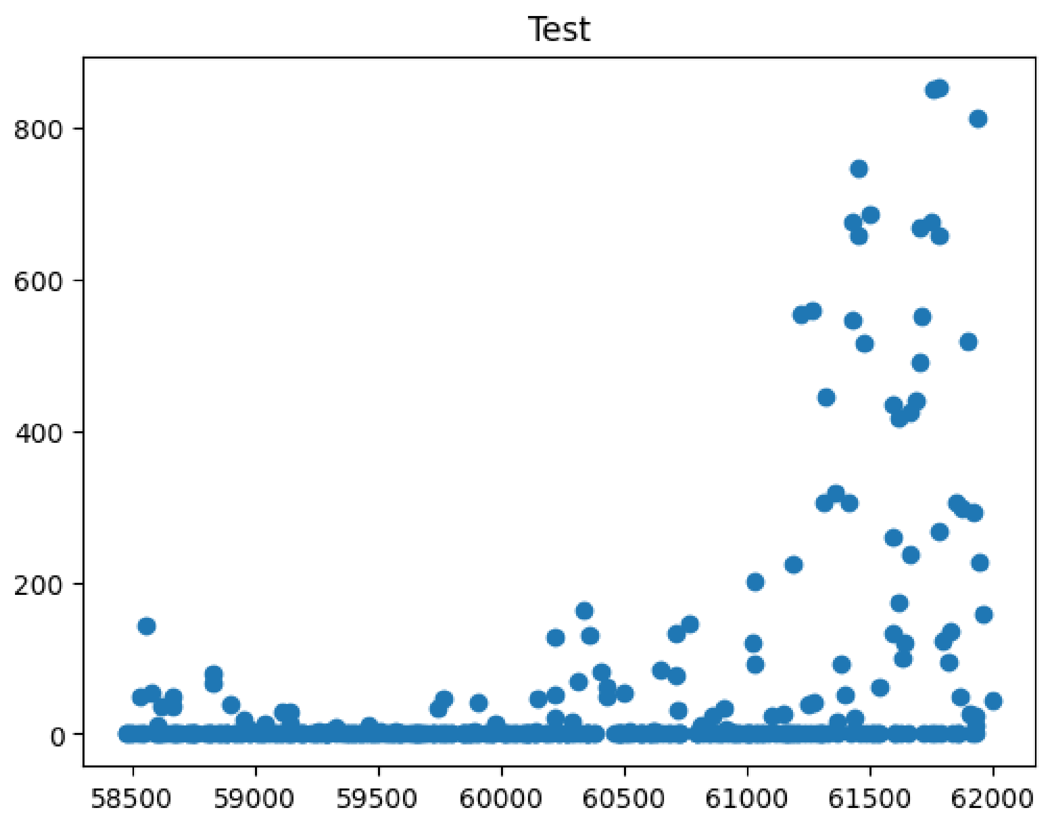
```
ParallelLocalFoldFittingStrategy
	-14.6686	 = Validation score   (-mean_absolute_error)
	33.25s	 = Training   runtime
	14.63s	 = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 525.56s of
the 525.56s of remaining time.
	-16.2298	 = Validation score   (-mean_absolute_error)
	6.74s	 = Training   runtime
	0.92s	 = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 517.04s of the
517.04s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-14.9675	 = Validation score   (-mean_absolute_error)
	199.96s	 = Training   runtime
	0.09s	 = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 315.82s of the
315.82s of remaining time.
	-15.393	 = Validation score   (-mean_absolute_error)
	1.41s	 = Training   runtime
	0.92s	 = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 312.6s of the
312.6s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-13.3296	 = Validation score   (-mean_absolute_error)
	34.83s	 = Training   runtime
	0.43s	 = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 276.1s of the 276.1s
of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-14.6344	 = Validation score   (-mean_absolute_error)
	87.17s	 = Training   runtime
	8.0s	 = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 184.74s of the
184.74s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-12.8806	 = Validation score   (-mean_absolute_error)
	144.34s	 = Training   runtime
	0.41s	 = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 38.92s of the
38.92s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-14.13	 = Validation score   (-mean_absolute_error)
	33.23s	 = Training   runtime
```

```
        6.02s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
1.28s of remaining time.
        -12.4906         = Validation score   (-mean_absolute_error)
        0.42s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 599.16s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_121_B/")
Evaluation: mean_absolute_error on test data: -10.565288276219954
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -10.565288276219954,
    "root_mean_squared_error": -28.9004043053015,
    "mean_squared_error": -835.2333690098895,
    "r2": 0.9640423931350784,
    "pearsonr": 0.9818710215871315,
    "median_absolute_error": -0.5272443890571594
}

Evaluation on test data:
-10.565288276219954
```




```
                     model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -10.565288 -12.490623        1.653043      16.924535
211.994505                 0.004068                0.000609           0.418880
2        True         12
1    NeuralNetTorch_BAG_L1  -10.922194 -12.880574        0.202651       0.413828
144.337203                 0.202651                0.413828         144.337203
1        True         10
2          LightGBM_BAG_L1  -11.107095 -14.668604        0.877050      14.628683
33.254581                0.877050               14.628683          33.254581
1        True          4
3        LightGBMXT_BAG_L1  -11.250137 -13.568253        0.904221      15.166392
31.000424                0.904221               15.166392          31.000424
1        True          3
4     LightGBMLarge_BAG_L1  -11.290533 -14.129996        1.138302       6.016109
33.229313                1.138302                6.016109          33.229313
1        True         11
5          CatBoost_BAG_L1  -11.580680 -14.967494        0.064644       0.086183
199.957813                 0.064644                0.086183         199.957813
1        True          6
```

```
6           XGBoost_BAG_L1  -11.726623 -14.634374        1.342632       8.003729
87.166754                 1.342632                8.003729          87.166754
1        True           9
7    NeuralNetFastAI_BAG_L1  -12.677632 -13.329554        0.172148       0.426429
34.830028                 0.172148                0.426429          34.830028
1        True           8
8    RandomForestMSE_BAG_L1  -12.781962 -16.229756        0.389215       0.915921
6.738379                 0.389215                0.915921          6.738379
1        True           5
9      ExtraTreesMSE_BAG_L1  -13.117027 -15.392978        0.369955       0.917277
1.407970                 0.369955                0.917277          1.407970
1        True           7
10    KNeighborsDist_BAG_L1  -23.570593 -28.797984        0.017738       0.355572
0.028830                 0.017738                0.355572          0.028830
1        True           2
11    KNeighborsUnif_BAG_L1  -24.697224 -28.544405        0.019821       0.339523
0.028523                 0.019821                0.339523          0.028523
1        True           1
```



[21]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

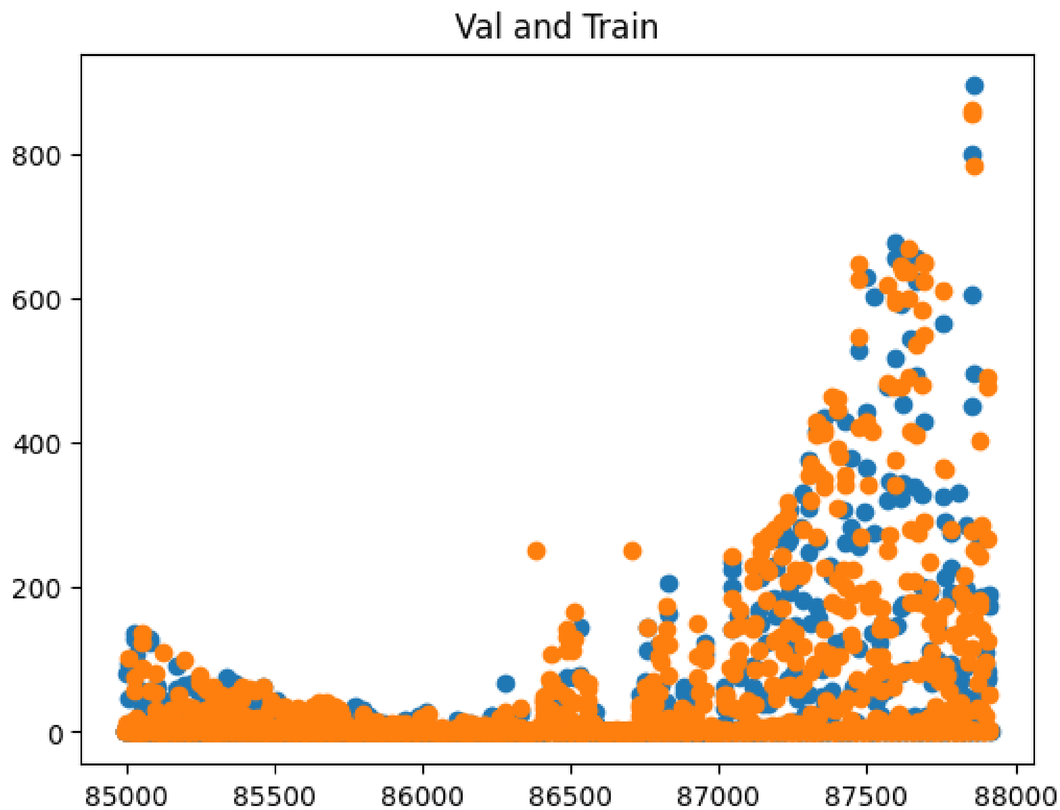
```
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Training model for location C…

Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_121_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   161.25 GB / 315.93 GB (51.0%)
Train Data Rows:    24073
Train Data Columns: 44
Tuning Data Rows:    1481
Tuning Data Columns: 44
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, -0.0, 80.87539, 169.67845)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    129804.36 MB
        Train Data (Original)  Memory Usage: 10.27 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 2 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 3): ['elevation:m', 'snow_drift:idx',
'location']
                These features carry no predictive signal and should be manually
```

```
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 40 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 39 | ['absolute_humidity_2m:gm3',
'air_density_2m:kgm3', 'ceiling_height_agl:m', 'clear_sky_energy_1h:J',
'clear_sky_rad:W', …]
                ('int', ['bool']) :  2 | ['snow_density:kgm3', 'is_estimated']
        0.1s = Fit runtime
        41 features in original data used to generate 41 features in processed
data.
        Train Data (Processed) Memory Usage: 8.02 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.17s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
```

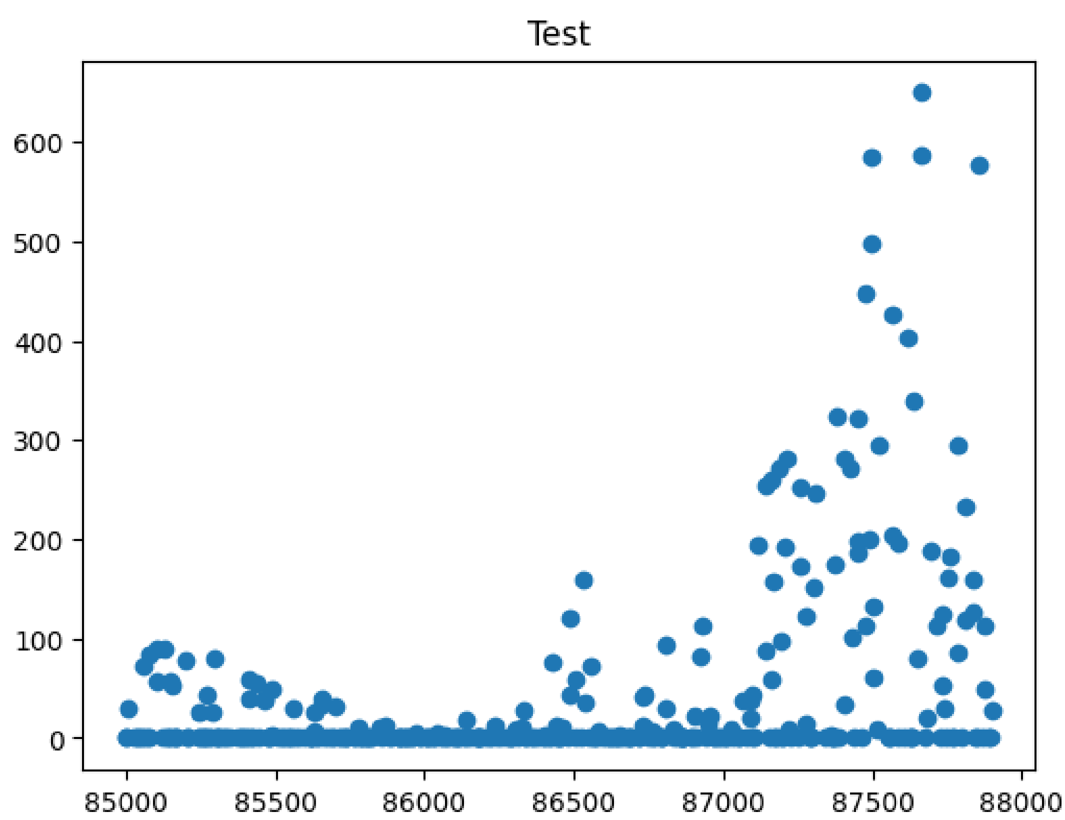
```
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.83s of the
599.83s of remaining time.
        -19.8149         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        1.21s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 598.4s of the
598.39s of remaining time.
        -20.1923         = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.26s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 598.04s of the
598.04s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -11.8238         = Validation score   (-mean_absolute_error)
        30.5s    = Training   runtime
        12.78s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 563.13s of the
563.13s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.8555         = Validation score   (-mean_absolute_error)
        32.42s   = Training   runtime
        11.07s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 526.18s of
the 526.17s of remaining time.
        -16.5945         = Validation score   (-mean_absolute_error)
        5.61s    = Training   runtime
        0.76s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 519.24s of the
519.24s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.2021         = Validation score   (-mean_absolute_error)
        198.59s  = Training   runtime
        0.07s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 319.43s of the
319.43s of remaining time.
        -15.4038         = Validation score   (-mean_absolute_error)
        1.16s    = Training   runtime
        0.78s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 316.87s of
the 316.87s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.5826         = Validation score   (-mean_absolute_error)
        30.89s   = Training   runtime
        0.39s    = Validation runtime
```

```
Fitting model: XGBoost_BAG_L1 … Training model for up to 284.49s of the
284.48s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.135  = Validation score   (-mean_absolute_error)
        57.71s   = Training   runtime
        3.44s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 223.32s of the
223.31s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.6436         = Validation score   (-mean_absolute_error)
        80.9s    = Training   runtime
        0.32s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 141.05s of the
141.05s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.9072         = Validation score   (-mean_absolute_error)
        103.55s  = Training   runtime
        11.87s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
28.52s of remaining time.
        -11.6344         = Validation score   (-mean_absolute_error)
        0.41s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 572.36s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_121_C/")
Evaluation: mean_absolute_error on test data: -12.106373514343234
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -12.106373514343234,
    "root_mean_squared_error": -29.628292635979346,
    "mean_squared_error": -877.8357245232279,
    "r2": 0.9110192456629367,
    "pearsonr": 0.9568709563350682,
    "median_absolute_error": -0.7409093677997589
}

Evaluation on test data:
-12.106373514343234
```




```
                       model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -12.106374 -11.634406        1.280072      14.697230
142.728171                 0.003852                0.000596           0.413717
2        True         12
1        LightGBMXT_BAG_L1  -12.167087 -11.823833        0.896617      12.782660
30.503842                 0.896617               12.782660          30.503842
1        True          3
2    NeuralNetFastAI_BAG_L1  -13.232373 -13.582617        0.172593       0.386044
30.885467                 0.172593                0.386044          30.885467
1        True          8
3     NeuralNetTorch_BAG_L1  -13.306948 -13.643575        0.191212       0.316859
80.897488                 0.191212                0.316859          80.897488
1        True         10
4          LightGBM_BAG_L1  -13.515471 -12.855477        0.880363      11.074447
32.418326                 0.880363               11.074447          32.418326
1        True          4
5          CatBoost_BAG_L1  -13.547169 -13.202079        0.071552       0.073299
198.594900                 0.071552                0.073299         198.594900
1        True          6
```

```
6            XGBoost_BAG_L1   -13.960029 -13.134987        0.731192       3.441086
57.707493                 0.731192                3.441086           57.707493
1        True           9
7      LightGBMLarge_BAG_L1   -14.282340 -12.907218        2.930983      11.868326
103.548933                 2.930983               11.868326          103.548933
1        True          11
8      ExtraTreesMSE_BAG_L1   -15.433922 -15.403829        0.278085       0.775379
1.158327                 0.278085               0.775379           1.158327
1        True           7
9    RandomForestMSE_BAG_L1   -16.828232 -16.594465        0.255087       0.762795
5.610942                 0.255087               0.762795           5.610942
1        True           5
10   KNeighborsUnif_BAG_L1   -20.049167 -19.814903        0.015798       1.211072
0.027658                 0.015798               1.211072           0.027658
1        True           1
11   KNeighborsDist_BAG_L1   -20.130194 -20.192291        0.010560       0.260093
0.027700                 0.010560               0.260093           0.027700
1        True           2
```



[22]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

```
for i in range(len(predictors)):
    print(f"Predictor {i}:")
    print(predictors[i].
 ↪info()["model_info"]["WeightedEnsemble_L2"]["children_info"]["S1F1"]["model_weights"])
```

```
Predictor 0:
{'LightGBMXT_BAG_L1': 0.3409090909090909, 'NeuralNetTorch_BAG_L1':
0.38636363636363635, 'LightGBMLarge_BAG_L1': 0.2727272727272727}
Predictor 1:
{'LightGBMXT_BAG_L1': 0.3050847457627119, 'ExtraTreesMSE_BAG_L1':
0.03389830508474576, 'NeuralNetFastAI_BAG_L1': 0.22033898305084745,
'NeuralNetTorch_BAG_L1': 0.4406779661016949}
Predictor 2:
{'KNeighborsUnif_BAG_L1': 0.06451612903225808, 'LightGBMXT_BAG_L1':
0.7634408602150539, 'NeuralNetFastAI_BAG_L1': 0.021505376344086027,
'NeuralNetTorch_BAG_L1': 0.15053763440860218}
```

5 Submit

[23]:
```
import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

```
Loaded data from: X_test_raw.csv | Columns = 45 / 45 | Rows = 4608 -> 4608
```

[24]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
 ↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

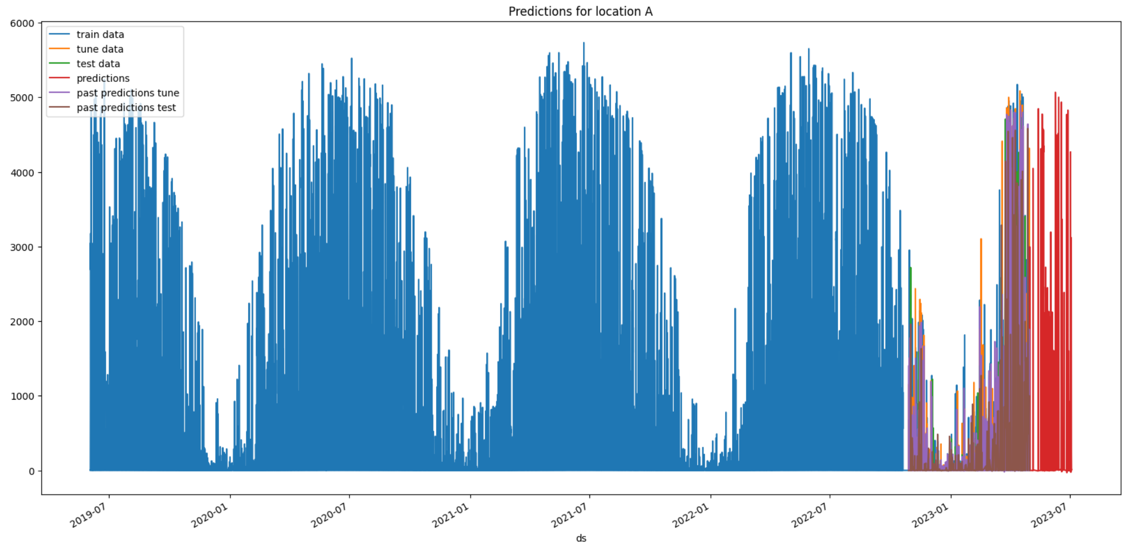
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

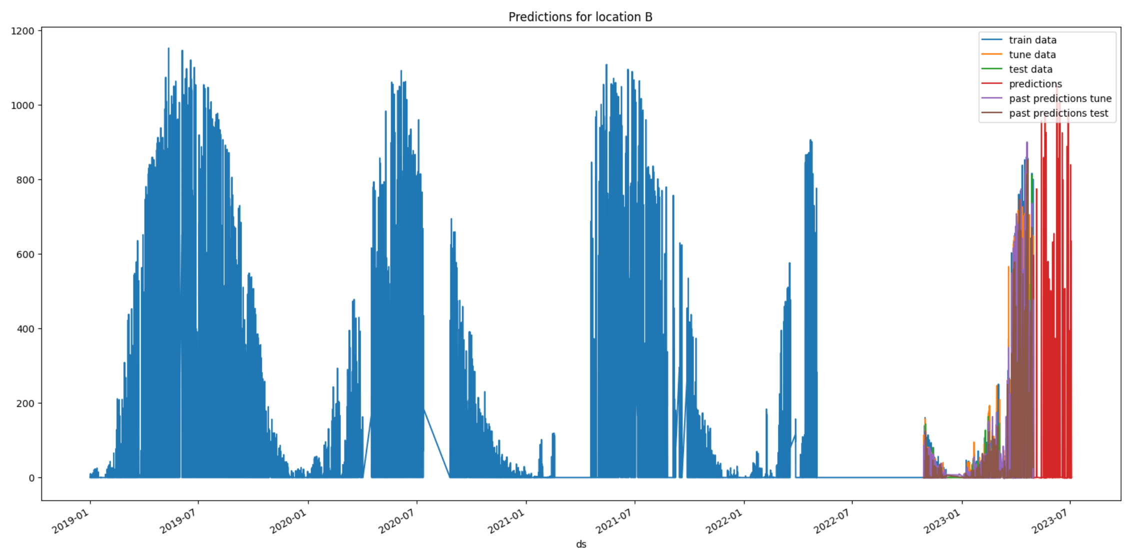
[25]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
```

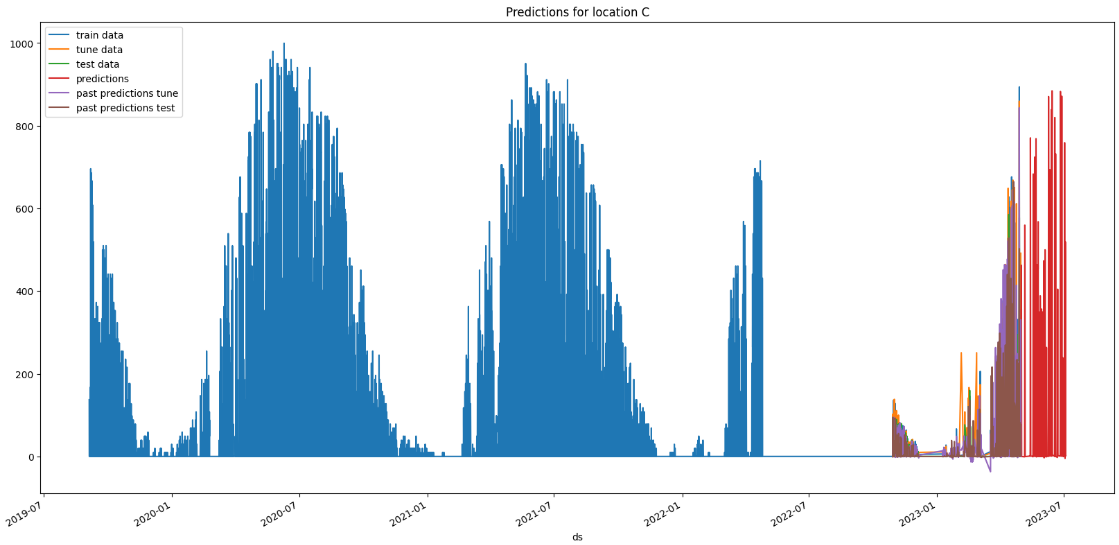
```
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(test_data[test_data["location"] == loc])
```

[26]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions test")
```

```python
# title
ax.set_title(f"Predictions for location {loc}")
```








```
[27]: temp_predictions = [prediction.copy() for prediction in predictions]
      if clip_predictions:
          # clip predictions smaller than 0 to 0
          for pred in temp_predictions:
              # print smallest prediction
              print("Smallest prediction:", pred["prediction"].min())
              pred.loc[pred["prediction"] < 0, "prediction"] = 0
              print("Smallest prediction after clipping:", pred["prediction"].min())

      # Instead of clipping, shift all prediction values up by the largest negative␣
       ↪number.
      # This way, the smallest prediction will be 0.
      elif shift_predictions:
          for pred in temp_predictions:
              # print smallest prediction
              print("Smallest prediction:", pred["prediction"].min())
              pred["prediction"] = pred["prediction"] - pred["prediction"].min()
              print("Smallest prediction after clipping:", pred["prediction"].min())

      elif shift_predictions_by_average_of_negatives_then_clip:
          for pred in temp_predictions:
              # print smallest prediction
              print("Smallest prediction:", pred["prediction"].min())
              mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
              # if not nan
              if mean_negative == mean_negative:
                  pred["prediction"] = pred["prediction"] - mean_negative
```

```
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
Smallest prediction: -31.128567
Smallest prediction after clipping: 0.0
Smallest prediction: -2.1167367
Smallest prediction after clipping: 0.0
Smallest prediction: -4.853957
Smallest prediction after clipping: 0.0
```

[27]:

```
       id  prediction
0       0    0.000000
1       1    0.000000
2       2    0.000000
3       3   30.958389
4       4  311.790527
..    …          …
715  2155   66.551102
716  2156   39.532722
717  2157    8.220373
718  2158    1.611563
719  2159    1.520409

[2160 rows x 2 columns]
```

[28]:

```
# Save the submission DataFrame to submissions folder, create new name based on␣
↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_121.csv
jall1a
```

[ ]:

```
# feature importance
# print starting calculating feature importance for location A with big text␣
↪font
```

```
print("\033[1m" + "Calculating feature importance for location A..." +
 ↪"\033[0m")
predictors[0].feature_importance(feature_stage="original",
 ↪data=test_data[test_data["location"] == "A"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location B..." +
 ↪"\033[0m")
predictors[1].feature_importance(feature_stage="original",
 ↪data=test_data[test_data["location"] == "B"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location C..." +
 ↪"\033[0m")
predictors[2].feature_importance(feature_stage="original",
 ↪data=test_data[test_data["location"] == "C"], time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'snow_drift:idx', 'location', 'prediction']
Computing feature importance via permutation shuffling for 41 features using 361
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location A…

```
        1743.39s        = Expected runtime (174.34s per shuffle set)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),
 ↪"autogluon_each_location.ipynb"])
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

[ ]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,
 ↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
 ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."
```

```
# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```

[ ]:

```

```